CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: September 13, 2012

A. Castellani
University of Padova
S. Loreto
Ericsson
A. Rahman
InterDigital Communications, LLC
T. Fossati
KoanLogic
E. Dijk
Philips Research
March 12, 2012

Best practices for HTTP-CoAP mapping implementation
draft-castellani-core-http-mapping-03

Abstract

This draft aims at being a base reference documentation for HTTP-CoAP proxy implementors. It details deployment options, discusses possible approaches for URI mapping, and provides useful considerations related to protocol translation.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 13, 2012.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of

publication of this document. Please review these documents
carefully, as they describe your rights and restrictions with respect
to this document. Code Components extracted from this document must
include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.

Table of Contents

1. Introduction 4
2. Terminology 4
3. Cross-protocol resource identification using URIs 5
3.1. URI mapping 6
3.1.1. Homogeneous mapping 7
3.1.2. Embedded mapping 7
4. HTTP-CoAP implementation 8
4.1. Placement and deployment 8
4.2. Basic mapping 10
4.2.1. Caching and congestion control 11
4.2.2. Cache Refresh via Observe 12
4.2.3. Use of CoAP blockwise transfer 13
4.2.4. Use case: HTTP/IPv4-CoAP/IPv6 proxy 13
4.3. Multiple message exchanges mapping 15
4.3.1. Relevant features of existing standards 15
4.3.2. Multicast mapping 16
4.3.3. Multicast responses caching 19
4.3.4. Observe mapping 20
5. CoAP-HTTP implementation 28
5.1. Placement and Deployment 29
5.2. Basic mapping 30
5.2.1. Payloads and Media Types 30
5.2.2. Max-Age and ETag Options 31
5.2.3. Use of CoAP blockwise transfer 31
5.2.4. HTTP Status Codes 1xx and 3xx 31
5.2.5. Examples 31
6. Security Considerations 33
6.1. Traffic overflow 34
6.2. Cross-protocol security policy mapping 34
6.3. Handling secured exchanges 35
6.4. Spoofing and Cache Poisoning 35
6.5. Subscription 36
7. Acknowledgements 36
8. References 36
8.1. Normative References 36
8.2. Informative References 38
Appendix A. Internal Mapping Functions (from an implementer's perspective) 39
A.1. URL Map Algorithm 39
A.2. Security Policy Map Algorithm 40
A.3. Content-Type Map Algorithm 41
Authors' Addresses 41

1. Introduction

RESTful protocols, such as HTTP [RFC2616] and CoAP [I-D.ietf-core-coap], can interoperate through an intermediary proxy which performs cross-protocol mapping.

A reference about the mapping process is provided in Section 8 of [I-D.ietf-core-coap]. However, depending on the involved application, deployment scenario, or network topology, such mapping could be realized using a wide range of intermediaries.

Moreover, the process of implementing such a proxy could be complex, and details regarding its internal procedures and design choices deserve further discussion, which is provided in this document.

This draft is organized as follows:

- Section 2 describes terminology to identify different mapping approaches and the related proxy deployments;

- Section 3 discusses impact of the mapping on URI and describes notable options;

- Section 4 and Section 5 respectively analyze the mapping from HTTP to CoAP and viceversa;

- Section 6 discusses possible security impact related to the mapping.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Terminology

A device providing cross-protocol HTTP-CoAP mapping is called an HTTP-CoAP cross-protocol proxy (HC proxy).

At least two different kinds of HC proxies exist:

- One-way cross-protocol proxy (1-way proxy): This proxy translates from a client of a protocol to a server of another protocol but not vice-versa.

- Two-way (or bidirectional) cross-protocol proxy (2-way proxy): This proxy translates from a client of both protocols to a server supporting one protocol.

1-way and 2-way HC proxies are realized using the following general types of proxies:

Forward proxy (F): Is a proxy known by the client (either CoAP or HTTP) used to access a specific cross-protocol server (respectively HTTP or CoAP). Main feature: server(s) do not require to be known in advance by the proxy (ZSC: Zero Server Configuration).

Reverse proxy (R): Is a proxy known by the client to be the server, however for a subset of resources it works as a proxy, by knowing the real server(s) serving each resource. When a cross-protocol resource is accessed by a client, the request will be silently forwarded by the reverse proxy to the real server (running a different protocol). If a response is received by the reverse proxy, it will be mapped, if possible, to the original protocol and sent back to the client. Main feature: client(s) do not require to know in advance the proxy (ZCC: Zero Client Configuration).

Interception proxy (I): This proxy [RFC3040] can intercept any origin protocol request (HTTP or CoAP) and map it to the destination protocol, without any kind of knowledge about the client or server involved in the exchange. Main feature: client(s) and server(s) do not require to know or be known in advance by the proxy (ZCC and ZSC).

A server-side (SS) proxy is placed in the same network domain of the server; Conversely a client-side (CS) is in the same network domain of the client. Differently from these two cases, the proxy is said to be External (E).

3. Cross-protocol resource identification using URIs

A Uniform Resource Identifier (URI) provides a simple and extensible means for identifying a resource. It enables uniform identification of resources via a separately defined extensible set of naming schemes [RFC3986].

URIs are formed of at least three components: scheme, authority and path. The scheme is the first part of the URI, and it often corresponds to the protocol used to access the resource. However, as noted in Section 1.2.2 of [RFC3986] the scheme does not imply that a particular protocol is used to access the resource.

Clients using URIs to identify target resources (e.g. HTTP web browsers) may support only a limited set of schemes (i.e. 'http',

'https'). If such clients need to interoperate with resources identified by an unsupported scheme (e.g. 'coap'), the existence of a URI using a scheme supported by the client is required for interoperability.

Both CoAP and HTTP implement the REST paradigm, so, in principle, the same resource can be made available in each protocol if protocol translation is applied.

In general two different procedures can be used by a client to access cross-protocol resources:

Protocol-aware access: Happens when a client knows the other protocol domain and accesses the cross-protocol resource using its URI by using an HC proxy. Example: An HTTP client accesses a CoAP resource by addressing it using the 'coap' scheme inside the HTTP request (actual protocol translation is provided by the proxy). Both HTTP and CoAP allow using different schemes in requests to proxies: (i) see Section 4.1.2 of [I-D.ietf-httpbis-p1-messaging], and (ii) Section 5.10.3 of [I-D.ietf-core-coap].

Protocol-agnostic access: The client accesses the cross-protocol resource using an URI with a scheme supported by the client (e.g. uses 'http' scheme to access a CoAP resource), URI and protocol translation is provided by a cross-protocol proxy. In order to use this method a URI identifying an equivalent resource MUST exist, and SHOULD be provided by the cross-protocol proxy.

URI mapping is NOT required when using protocol-aware access, the following section is focused on URI mapping techniques for protocol-agnostic access.

3.1. URI mapping

When accessing cross-protocol resources in a protocol-agnostic way, clients MUST use an URI with a scheme supported by the client.

Since determination of equivalence of URIs (e.g. whether or not they identify the same resource) is based on lexicographic comparison, URI domains using different schemes are fully distinct: resources identified by the same authority and path tuple change when switching the scheme.

Example: Assume that the following resource exists - "coap://node.coap.something.net/foo". The resource identified by "http://node.coap.something.net/foo" may not exist or be non-equivalent to the one identified by the 'coap' scheme.

If a cross-protocol URI exists providing an equivalent representation of the native protocol resource, it can be provided by a different URI (in terms of authority and path). The mapping of an URI between HTTP and CoAP is said HC URI mapping.

Example: The HC URI mapping to HTTP of the CoAP resource identified by "coap://node.coap.something.net/foo" is "http://node.something.net/foobar".

The process of providing the HC URI mapping could be complex, since a proper mechanism to statically or dynamically (discover) map the resource HC URI mapping is required.

Two static HC URI mappings are discussed in the following subsections.

3.1.1. Homogeneous mapping

The URI mapping between CoAP and HTTP is called homogeneous, if the same resource is identified by URIs with different schemes.

Example: The CoAP resource "//node.coap.something.net/foo" identified either by the URI "coap://node.coap.something.net/foo", and or by the URI "http://node.coap.something.net/foo" is the same. When the resource is accessed using HTTP, the mapping from HTTP to CoAP is performed by an HC proxy

When homogeneous HC URI mapping is available, HC-I proxies are easily implementable.

3.1.2. Embedded mapping

When the HC URI mapping of the resource embeds inside it the authority and path part of the native URI, then the mapping is said to be embedded.

Example: The CoAP resource "coap://node.coap.something.net/foo" can be accessed at "http://hc-proxy.something.net/coap/node.coap.something.net/foo".

This mapping technique can be used to reduce the mapping complexity in an HC reverse proxy.

3.1.2.1. HTML5 scheme handler registration

The draft HTML5 standard offers a mechanism that allows an HTTP user agent to register a custom scheme handler through an HTML5 web page. This feature permits to an HC proxy to be registered as "handler" for

URIs with the 'web+coap' or 'web+coaps' schemes using an HTML5 web page which embeds the custom scheme handler registration call registerProtocolHandler() described in Section 6.5.1.2 of [W3C.HTML5].

Example: the HTML5 homepage of a HC proxy at h2c.example.org could include the method call:

```
registerProtocolHandler('web+coap', 'proxy?url=%s', 'example HC proxy')
```

This registration call will prompt the HTTP user agent to ask for the user's permission to register the HC proxy as a handler for all 'web+coap' URIs. If the user accepts, whenever a 'web+coap' link is requested, the request will be fulfilled through the HC proxy: URI "web+coap://foo.org/a" will be transformed into URI "http://h2c.example.org/proxy?url=web+coap://foo.org/a".

4. HTTP-CoAP implementation

4.1. Placement and deployment

In typical scenarios the HC proxy is expected to be server-side (SS), in particular deployed at the edge of the constrained network.

The arguments supporting SS placement are the following:

TCP/UDP: Translation between HTTP and CoAP requires also a TCP to UDP mapping; UDP performance over the unconstrained Internet may not be adequate. In order to minimize the number of required retransmissions and overall reliability, TCP/UDP conversion SHOULD be performed at a SS placed proxy.

Caching: Efficient caching requires that all the CoAP traffic is intercepted by the same proxy, thus an SS placement, collecting all the traffic, is strategical for this need.

Multicast: To support using local-multicast functionalities available in the constrained network, the HC proxy MAY require a network interface directly attached to the constrained network.

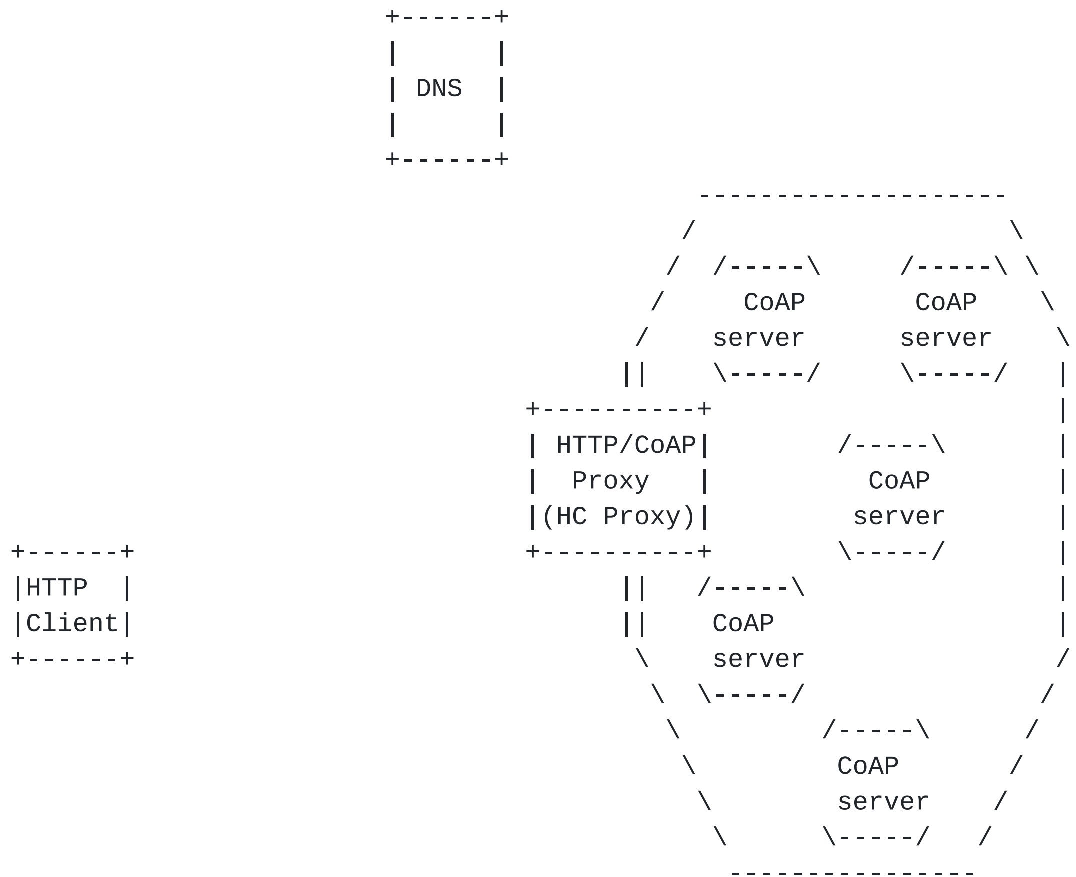

Figure 1: Server-side HC proxy deployment scenario

Other important aspects involved in the selection of which type of proxy deployment, whose choice impacts its placement too, are the following:

Client/Proxy/Network configuration overhead: Forward proxies require either static configuration or discovery support in the client. Reverse proxies require either static configuration, server discovery or embedded URI mapping in the proxy. Interception proxies require minimal deployment effor (i.e. web traffic routing towards the proxy).

Scalability/Availability: Both aspects are typically addressed using redundancy. CS deployments, due to the limited catchment area and administrative-wide domain of operation, have looser requirements on this. SS deployments, in dense/popular/critical environments, have stricter requirements and MAY need to be replicated. Stateful proxies (e.g. reverse) may be complex to replicate.

Discussion about security impacts of different deployments is covered

in Section 6.

Table 1 shows some interesting HC proxy deployment scenarios, and notes the advantages related to each scenario.

Feature	F CS	R SS	I SS
TCP/UDP	-	+	+
Multicast	-	+	+
Caching	-	+	+
Scalability/Availability	+	+/-	+
Configuration	-	-	+

Table 1: Interesting HC proxy deployments

Guidelines proposed in the previous paragraphs have been used to fill out the above table. In the first three rows, it can be seen that SS deployment is preferred versus CS. Scalability/Availability issues can be generally handled, but some complexity may be involved in reverse proxies scenarios. Configuration overhead could be simplified when interception proxies deployments are feasible.

When support for legacy HTTP clients is required, it may be preferable using configuration/discovery free deployments. Discovery procedures for client or proxy auto-configuration are still under active-discussion: see [I-D.vanderstok-core-bc], [I-D.bormann-core-simple-server-discovery] or [I-D.shelby-core-resource-directory]. Static configuration of multiple forward proxies is typically not feasible in existing HTTP clients.

4.2. Basic mapping

The mapping of HTTP requests to CoAP and of the response back to HTTP is defined in Section 8.2 of [I-D.ietf-core-coap].

The mapping of a CoAP response code to HTTP is not straightforward, this mapping MUST be operated accordingly to Table 4 of [I-D.ietf-core-coap].

No temporal upper bound is defined for a CoAP server to provide the response, thus for long delays the HTTP client or any other proxy in between MAY timeout. Further discussion is available in Section 7.1.4 of [I-D.ietf-httpbis-p1-messaging].

The HC proxy MUST define an internal timeout for each pending CoAP

request, because the CoAP server may silently die before completing the request.

Even if the DNS protocol may not be used inside the constrained network, having valid DNS entries for constrained hosts, where possible, MAY help HTTP clients to access the resources offered by them.

An example of the usefulness of such entries is described in Section 4.2.4.

HTTP connection pipelining (section 7.1.2.2 of [I-D.ietf-httpbis-p1-messaging]) is transparent to the CoAP network: the HC proxy will sequentially serve the pipelined requests by issuing different CoAP requests.

4.2.1. Caching and congestion control

The HC proxy SHOULD limit the number of requests to CoAP servers by responding, where applicable, with a cached representation of the resource.

Duplicate idempotent pending requests to the same resource SHOULD in general be avoided, by duplexing the response to the relevant hosts without duplicating the request.

If the HTTP client times out and drops the HTTP session to the proxy (closing the TCP connection), the HC proxy SHOULD wait for the response and cache it if possible. Further idempotent requests to the same resource can use the result present in cache, or, if a response has still to come, requests will wait on the open CoAP session.

Resources experiencing a high access rate coupled with high volatility MAY be observed [I-D.ietf-core-observe] by the HC proxy to keep their cached representation fresh while minimizing the number of needed messages. See Section 4.2.2 for a heuristics that enables the HC proxy to decide whether observing is a more convenient strategy than ordinary refreshing via Max-Age/ETag-based mechanisms.

Specific deployments may show highly congested servers/resources -- e.g. multicast resources (see Section 4.3.2), popular servers, etc. A careful analysis is required to pick the correct caching policy involving these resources, also taking into consideration the security implications that may impact these targets specifically, and the constrained network in general.

To this end when traffic reduction obtained by the caching mechanism

is not adequate, the HC proxy could apply stricter policing by limiting the amount of aggregate traffic to the constrained network. In particular, the HC proxy SHOULD pose a rigid upper limit to the number of concurrent CoAP request pending on the same constrained network; further request MAY either be queued or dropped. In order to efficiently apply this congestion control, the HC proxy SHOULD be SS placed.

Further discussion on congestion control can be found in [I-D.eggert-core-congestion-control].

4.2.2. Cache Refresh via Observe

There are cases where using CoAP observe protocol to handle proxy cache refresh may be preferable to the validation mechanism based on ETag's defined in section 5.6.2 of [I-D.ietf-core-coap]. Such scenarios include, but are not limited to, sleeping nodes -- with possibly high variance in requests' distribution -- which would greatly benefit from a server driven cache update mechanism. Ideal candidates would also be the crowded or very low throughput networks, where reduction of the total number of exchanged messages is an important requirement.

This subsection aims at providing a practical evaluation method to decide whether the refresh of a cached resource R is more efficiently handled via ETag validation or by establishing an observation on R.

Let T_R be the mean time between two client requests to resource R, let F_R be the freshness lifetime of R representation, and let M_R be the total number of messages exchanged towards resource R. If we assume that the initial cost for establishing the observation is negligible, an observation on R reduces M_R iff T_R < 2*F_R with respect to using ETag validation, that is iff the mean arrival time of requests for resource R is greater than half the refresh rate of R.

When using observations M_R is always upper bounded by 2*F_R: in the constrained network no more than 2*F_R messages will be generated towards resource R.

Proof: Let T be the evaluated interval of time, let M_Ro be the total number of messages exchanged towards resource R using observation, and let M_Re be the total number of messages exchanged towards resource R using ETag validation. The following equations hold M_Re = T*2/T_R, M_Ro = T/F_R. M_Ro < M_Re iff 1/F_R < 2/T_R, that is T_R < 2*F_R. The amount of messages saved using observation is T*(2*F_R-T_R)/(T_R*F_R).

Example: assume that F_R is one second and T_R is 1.5 seconds. Since 1.5 is lower than 2*1, an observation on R reduces M_R. In a single day of usage, 28800 messages will be saved if the HC proxy establishes an observation on R. The single message cost required to establish this observation is negligible.

4.2.3. Use of CoAP blockwise transfer

An HC proxy SHOULD support CoAP blockwise transfers [I-D.ietf-core-block] to allow transport of large CoAP payloads while avoiding link-layer fragmentation in LLNs, and to cope with small datagram buffers in CoAP end-points as described in [I-D.ietf-core-coap]. An HC proxy SHOULD attempt to retry a CoAP PUT or POST request with a payload using blockwise transfer if the destination CoAP server responded with 4.13 (Request Entity Too Large) to the original request. An HC proxy SHOULD attempt to use blockwise transfer when sending a CoAP PUT or POST request message that is larger than BLOCKWISE_THRESHOLD. The value of BLOCKWISE_THRESHOLD is implementation-specific, for example it may set by an administrator, preset to a known or typical UDP datagram buffer size for CoAP end-points, to N times the size of a link-layer frame where e.g. N=5, preset to a known IP MTU value, or set to a known Path MTU value.

For improved latency an HC proxy MAY initiate a blockwise CoAP request triggered by an incoming HTTP request even when the HTTP request message has not yet been fully received, but enough data has been received to send one or more data blocks to a CoAP server already.

4.2.4. Use case: HTTP/IPv4-CoAP/IPv6 proxy

This section covers the expected common use case regarding an HTTP/ IPv4 client accessing a CoAP/IPv6 resource.

While HTTP and IPv4 are today widely adopted communication protocols in the Internet, a pervasive deployment of constrained nodes exploiting the IPv6 address space is expected: enabling direct interoperability of such technologies is a valuable goal.

An HC proxy supporting IPv4/IPv6 mapping is said to be a v4/v6 proxy.

An HC v4/v6 proxy SHOULD always try to resolve the URI authority, and SHOULD prefer using the IPv6 resolution if available. The authority part of the URI is used internally by the HC proxy and SHOULD not be mapped to CoAP.

Figure 2 shows an HTTP client on IPv4 (C) accessing a CoAP server on

IPv6 (S) through an HC proxy on IPv4/IPv6 (P). The DNS has an A record for "node.coap.something.net" resolving to the IPv4 address of the HC proxy, and an AAAA record with the IPv6 address of the CoAP server.

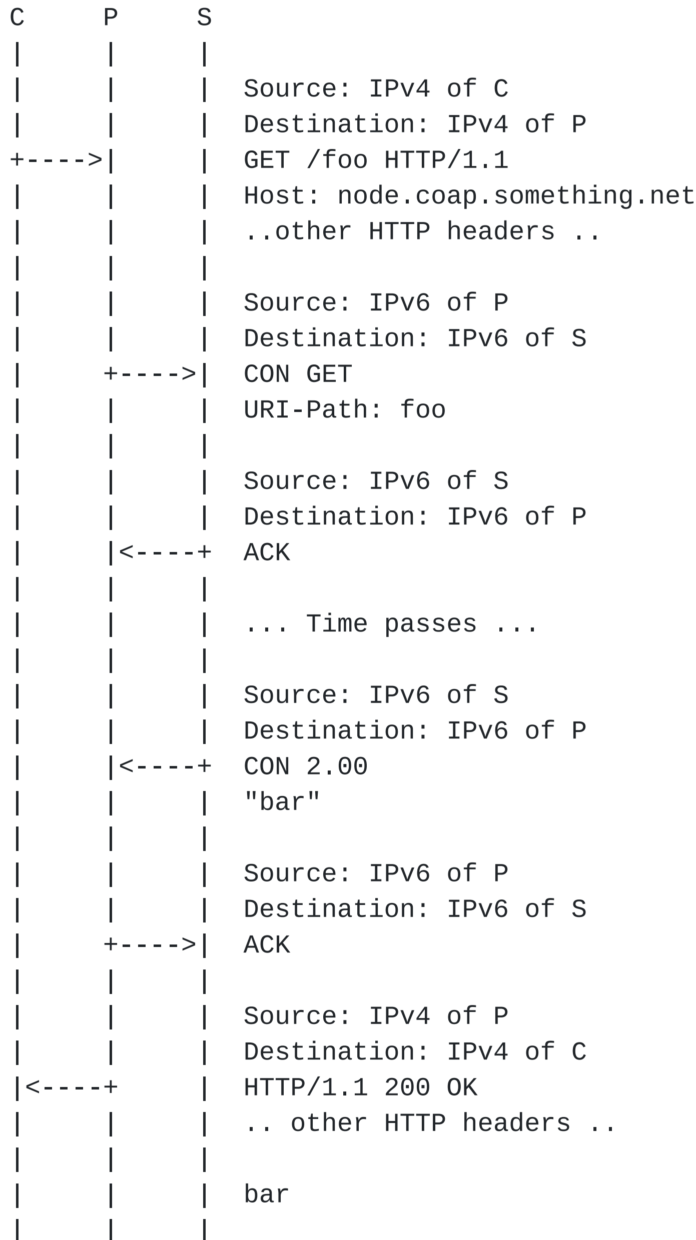

Figure 2: HTTP/IPv4 to CoAP/IPv6 mapping

The proposed example shows the HC proxy operating also the mapping between IPv4 to IPv6 using the authority information available in any HTTP 1.1 request. This way, IPv6 connectivity is not required at the

HTTP client when accessing a CoAP server over IPv6 only, which is a typical expected use case.

When P is an interception HC proxy, the CoAP request SHOULD have the IPv6 address of C as source (IPv4 can always be mapped into IPv6).

The described solution takes into account only the HTTP/IPv4 clients accessing CoAP/IPv6 servers; this solution does not provide a full fledged mapping from HTTP to CoAP.

In order to obtain a working deployment for HTTP/IPv6 clients, a different HC proxy access method may be required, or Internet AAAA records should not point to the node anymore (the HC proxy should use a different DNS database pointing to the node).

When an HC interception proxy deployment is used this solution is fully working even with HTTP/IPv6 clients.

4.3. Multiple message exchanges mapping

This section discusses the mapping of the multicast and observe features of CoAP, which have no corresponding primitive in HTTP, and as such are not immediately translatable.

The mapping, which must be considered in both the arrow directions (H->C, C->H) may involve multi-part responses, as in the multicast use case, asynchronous delivery through HTTP bidirectional techniques, and HTTP Web Linking in order to reduce the semantics lost in the translation.

4.3.1. Relevant features of existing standards

Various features provided by existing standards are useful to efficiently represent sessions involving multiple messages.

4.3.1.1. Multipart messages

In particular, the "multipart/*" media type, defined in Section 5.1 of [RFC2046], is a suitable solution to deliver multiple CoAP responses within a single HTTP payload. Each part of a multipart entity SHOULD be represented using "message/http" media type containing the full mapping of a single CoAP response as previously described.

4.3.1.2. Immediate message delivery

An HC proxy may prefer to transfer each CoAP response immediately after its reception. This is possible thanks to the HTTP Transfer-

Encoding "chunked", that enables transferring single responses without any further delay.

A detailed discussion on the use of chunked Transfer-Encoding to stream data over HTTP can be found in [RFC6202]. Large delays between chunks can lead the HTTP session to timeout, more details on this issue can be found in [I-D.thomson-hybi-http-timeout].

An HC proxy MAY prefer (e.g. to avoid buffering) to transfer each response related to a multicast request as soon as it comes in from the server. One possible way to achieve this result is using the "chunked" Transfer-Encoding in the HTTP response, to push individual responses until some trigger is fired (timeout, max number of messages, etc.).

An example showing immediate delivery of CoAP responses using HTTP chunks will be provided in Section 4.3.4, while describing its application to an observe session.

4.3.1.3. Detailing source information

Under some circumstances, responses may come from different sources (i.e. responses to a multicast request); in this case details about the actual source of each CoAP response MAY be provided to the client. Source information can be represented using HTTP Web Linking as defined in [RFC5988], by adding the actual source URI into each response using Link option with "via" relation type.

4.3.2. Multicast mapping

In order to establish a multicast communication such a feature should be offered either by the network (i.e. IP multicast, link-layer multicast, etc.) or by a gateway (i.e. the HC proxy). Rationale on the methods available to obtain such a feature is out-of-scope of this document, and extensive discussion of group communication techniques is available in [I-D.ietf-core-groupcomm].

Additional considerations related to handling multicast requests mapping are detailed in the following sections.

4.3.2.1. URI identification and mapping

In order to successfully handle a multicast request, the HC proxy MUST successfully perform the following tasks on the URI:

Identification: The HC proxy MUST understand whether the requested URI identifies a group of nodes.

Mapping: The HC proxy MUST know how to distribute the multicast request to involved servers; this process is specific of the group communication technology used.

When using IPv6 multicast paired with DNS, the mapping to IPv6 multicast is simply done using DNS resolution. If the group management is performed at the proxy, the URI or part of it (i.e. the authority) can be mapped using some static or dynamic table available at the HC proxy. In Section 3.5 of [I-D.ietf-core-groupcomm] discusses a method to build and maintain a local table of multicast authorities.

4.3.2.2. Request handling

When the HC proxy receives a request to a URI that has been successfully identified and mapped to a group of nodes, it SHOULD start a multicast proxying operation, if supported by the proxy.

Multicast request handling consists of the following steps:

Multicast TX: The HC proxy sends out the request on the CoAP side by using the methods offered by the specific group communication technology used in the constrained network;

Collecting RXs: The HC proxy collects every response related to the request;

Timeout: The HC proxy has to pay special attention in multicast timing, detailed discussion about timing depends upon the particular group communication technology used;

Distributing RXs to the client: The HC proxy can distribute the responses in two different ways: batch delivering them at the end of the process or on timeout, or immediately delivering them as they are available. Batch requires more caching and introduces delays but may lead to lower TCP overhead and simpler processing. Immediate delivery is the converse. A trade-off solution of partial batch delivery may also be feasible and efficient in some circumstances.

4.3.2.3. Examples

Figure 3 shows an HTTP client (C) requesting the resource "/foo" to a group of CoAP servers (S1/S2/S3) through an HC proxy (P) which uses IP multicast to send the corresponding CoAP request.

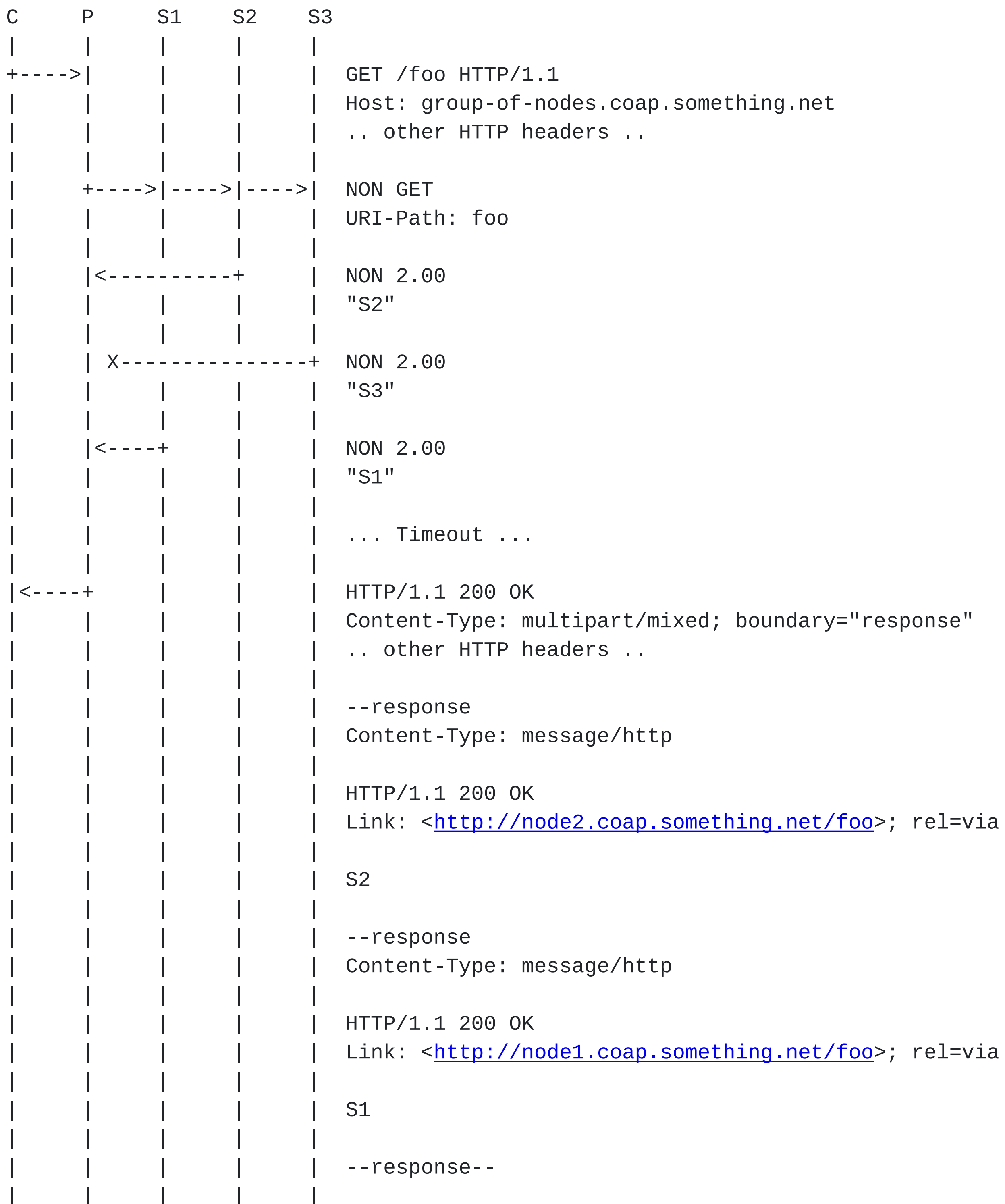

Figure 3: Unicast HTTP to multicast CoAP mapping

The example proposed in the above diagram does not make any assumption on which underlying group communication technology is

available in the constrained network. Some detailed discussion is provided about it along the following lines.

C makes a GET request to group-of-nodes.coap.something.net. This domain name MAY either resolve to the address of P, or to the IPv6 multicast address of the nodes (if IP multicast is supported and P is an interception proxy), or the proxy P is specifically known by the client that sends this request to it.

To successfully start multicast proxying operation, the HC proxy MUST know that the destination URI involves a group of CoAP servers, e.g. the authority group-of-nodes.coap.something.net is known to identify a group of nodes either by using an internal lookup table, using DNS paired with IPv6 multicast, or by using some other special technique.

A specific implementation option is proposed to further explain the proposed example. Assume that DNS is configured such that all subdomain queries to coap.something.net, such as group-of-nodes.coap.something.net, resolve to the address of P. P performs the HC URI mapping by removing the 'coap' subdomain from the authority and by switching the scheme from 'http' to 'coap' (result: "coap://group-of-node.something.net/foo"); "group-of-nodes.something.net" is resolved to an IPv6 multicast address to which S1, S2 and S3 belong. The proxy handles this request as multicast and sends the request "GET /foo" to the multicast group .

4.3.3. Multicast responses caching

We call perfect caching when the proxy uses only the cached representations to provide a response to the HTTP client. In the case of a multicast CoAP request, perfect caching is not adequate. This section updates the general caching guidelines of Section 4.2.1 with specific guidelines for the multicast use case.

Due to the inherent unreliable nature of the NON messages involved and since nodes may have dynamic membership in multicast groups, responding only with previously cached responses without issuing a new multicast request is not recommended. This perfect caching behaviour leads to miss responses of nodes that later joined the multicast group, and/or to repeately serve partial representations due to message losses. Therefore a multicast CoAP request SHOULD be sent by a HC proxy for each incoming request addressed to a multicast group.

Caching of multicast responses is still a valuable goal to pursue reduce network congestion, battery consumption and response latency. Some considerations to be performed when adopting a multicast caching behaviour are outlined in the following paragraph.

Caching of multicast GET responses MAY be implemented by adopting some technique that takes into account either knowledge about dynamic characteristics of group membership (occurrence or frequency of group changes) or even better its full knowledge (list of nodes currently part of the group).

When using a technique exploiting this knowledge, valid cached responses SHOULD be served from cache.

4.3.4. Observe mapping

By design, and certainly not without a good rationale, HTTP lacks a publish-subscriber facility. This implies that the mapping of the CoAP observe semantics has to be created ad hoc, perhaps by making use of one of the well-known HTTP techniques currently employed to establish an HTTP bidirectional connection with the target resource - as documented in [RFC6202].

In the following sections we will describe some of the approaches that can be used to identify an observable resource and to create the communication bridging needed to set up an end to end HTTP-CoAP observation.

4.3.4.1. Identification

In order to appropriately process an observe request, the HC proxy needs to know whether a given request is intended to establish an observation on the target resource, instead of triggering a regular request-response exchange.

At least two different approaches to identify such special requests exist, as discussed below.

4.3.4.1.1. Observable URI mapping

An URI is said to be observable whenever every request to it implicitly requires the establishment of an HTTP bidirectional connection to the resource.

Such subscription to the resource is always paired, if possible, to a CoAP observe session to the actual resource being observed. In general, multiple connections that are active with a single observable resource at the same time, are multiplexed to the single observe session opened by the intermediary. Its notifications are then de-multiplexed by the HC proxy to every HTTP subscriber.

An intermediary MAY pair a couple of distinct HTTP URIs to a single CoAP observable resource: one providing the usual request-response

mediated access to the resource, and the other that always triggers a CoAP observe session.

4.3.4.1.1.1. Discovery

As shown in Figure 4, in order to know whether an URI is observable, an HTTP UA MAY do a preflight request to the target resource using the HTTP OPTIONS method (see section 6.2 of [I-D.ietf-httpbis-p2-semantics]) to discover the communication options available for that resource.

If the resource supports observation, the proxy adds a Link Header [RFC5988] with the "obs" attribute as link-param (see Section 7 of [I-D.ietf-core-observe]).

```
C       P       S
|       |       |  OPTIONS /kitchen/temp HTTP/1.1
+------>|       |  Host: node.coap.something.net
|       |       |
|       +------>|  CON GET
|       |       |  Uri-Path: /.well-known/core?anchor=/kitchen/temp
|       |       |
|       |<------+  ACK 2.05
|       |       |  Payload: </kitchen/temp>;obs
|       |       |
|<------+       |  HTTP/1.1 200 OK
|       |       |  Link: </kitchen/temp>; obs; type="application/atom+xml"
|       |       |  Allow: GET, OPTIONS
```

Figure 4: Discover observability with HTTP OPTIONS

4.3.4.1.2. Differentiation using HTTP Header

Discerning an observation request through in-protocol means, e.g. via the presence and values of some HTTP metadata, avoids introducing static "observable" URIs in the HC proxy namespace. Though ideally the former should be preferred, there seems to be no standard way to use one of the established HTTP headers to convey the observe semantics.

Standardizing such methods is out-of-scope of this document, so we just point out some possible approaches that in the future may be used to differentiate observation requests from regular requests.

4.3.4.1.2.1. Expect Header

The first method involves the use of the Expect header as defined in Section 9.3 of [I-D.ietf-httpbis-p2-semantics]. Whenever an HC proxy receives a request with a "206-partial-content" expectation, the proxy MUST fulfill this expectation by pairing this request to either a new or existing observe session to the resource.

If the proxy is unable to observe the resource, or if the observation establishment fails, the proxy MUST reply to the client with "417 Expectation Failed" status code.

Given that the Expect header is processed hop-by-hop, this method will fail immediately in case a proxy not supporting this expectation is traversed. For this reason, at present, the said approach can't be used in the public Internet.

4.3.4.1.2.2. Prefer Header

A second, very similar, approach involves the use of the Prefer header, defined in [I-D.snell-http-prefer]. The HTTP user agent expresses the preference to establish an observation with the target resource by including a "streaming" preference to request an HTTP Streaming session, or a "long-polling" preference to signal to the proxy its intended polling behaviour (see [RFC6202]).

A compliant HC proxy will try to fulfill the preference, and manifest observation establishment success by responding with a status code of "206 Partial Content". The observation request fails, falling back to a single response, whenever the status code is different from 206.

This approach will never fail immediately, differently from the previous one, even across a chain of unaware proxies; however, as documented in [RFC6202], caching intermediaries may interfere, delay or block the HTTP bidirectional connection, making this approach unacceptable when no weak consistency of the resource can be tolerated by the requesting UA.

4.3.4.2. Notification(s) mapping

Multiplexing notifications using a single HTTP bidirectional session needs some further considerations about the selection of the media type that best fits this specific use case.

The usage of two different content-types that are suitable for carrying multiple notifications in a single session, is discussed in the following sections.

4.3.4.2.1. Multipart messaging

As already discussed in Section 4.3.1.1 for multicasting, the "multipart/*" media type is a suitable solution to deliver multiple CoAP notifications within a single HTTP payload.

As in the multicast case, each part of the multipart entity MAY be represented using a "message/http" media type, containing the full mapping of the single CoAP notification mapped, so that CoAP envelope informations are preserved (e.g. the response code).

A more sophisticated mapping could use multipart/mixed with native or translated media type.

4.3.4.2.2. Using ATOM Feeds

Popular observable resources with refresh rates higher than a couple of seconds may be treated as Atom feeds [RFC4287], especially with delay tolerant user agents and where persistence is required.

Figure 4 shows a resource supporting 'application/atom+xml' media-type. In such case clients can listen to update notification by regularly polling the resource via opportunely spaced GETs, i.e. driven by the advertised max-age value.

4.3.4.3. Examples

Figure 5 shows the interaction between an HTTP client (C), an HC proxy (P), and a CoAP server (S) for the observation of the resource "temperature" (T) available on S.

C manifests its intention to observe T by including the Expect Header in the request; if P or S do not support this interaction, the request MUST fail with "417 Expectation Failed" return code. In the presented example, both P and C support this interaction, and the subscription is successful, as stated by the "206 Partial Content" return code.

At every notification corresponds the emission of a HTTP chunk containing a single part, which contains a "message/http" payload containing the full mapping of the notification. When the observation is dropped by the CoAP server, the HTTP streaming session is closed.

```
C     P     S
|     |     |
+---->|     |  GET /temperature HTTP/1.1
```

```
    |     |     |  Host: node.coap.something.net
    |     |     |  Expect: 206-partial-content
    |     |     |  Accept: multipart/mixed
    |     |     |
    |     +---->|  CON GET
    |     |     |  Uri-Path: temperature
    |     |     |  Observe: 0
    |     |     |
    |     |<----+  ACK 2.05
    |     |     |  Observe: 3482
    |     |     |  "22.1 C"
    |     |     |
    |<----+     |  HTTP/1.1 206 Partial Content
    |     |     |  Content-Type: multipart/mixed; boundary=notification
    |     |     |
    |     |     |  XX
    |     |     |  --notification
    |     |     |  Content-Type: message/http
    |     |     |
    |     |     |  HTTP/1.1 200 OK
    |     |     |
    |     |     |  22.1 C
    |     |     |
    |     |     |  ... about 60 seconds have passed ...
    |     |     |
    |     |<----+  NON 2.05
    |     |     |  Observe: 3542
    |     |     |  "21.6 C"
    |     |     |
    |<----+     |  YY
    |     |     |  --notification
    |     |     |  Content-Type: message/http
    |     |     |
    |     |     |  HTTP/1.1 200 OK
    |     |     |
    |     |     |  21.6 C
    |     |     |
    |     |     |  ... if the server drops the relationship ...
    |     |     |
    |     |<----+  NON 2.05
    |     |     |  "21.8 C"
    |     |     |
    |<----+     |  ZZ
    |     |     |  --notification
    |     |     |  Content-Type: message/http
    |     |     |
    |     |     |  HTTP/1.1 200 OK
    |     |     |
```

```
|     |     |  21.8 C
|     |     |
|     |     |  --notification--
|     |     |
|     |     |  0
```

Figure 5: HTTP Streaming to CoAP Observe

Figure 6 shows the interaction between an HTTP client (C), an HC proxy (P), and a CoAP server (S) for the observation of the resource "temperature" (T) available on S.

C manifests its intention to observe T by including the Prefer Header in the request; if P or S do not support this interaction, the request silently fails if a status code "200 OK" is returned, which means that no further notification is expected on that session.

In the presented example, both P and C support this interaction, and the subscription is successful, as stated by the "206 Partial Content" status code. At every notification a new response is sent to the pending client, always containing the "206 Partial Content" status code, to indicate that the observe session is still active, so that C can issue a new long-polling request immediately after this notification.

If the observation relationship is dropped by S, P notifies the last received content using the "200 OK" status code, indicating that no further notification is expected on this observe session.

```
   C     P     S
   |     |     |
   +---->|     |  GET /temperature HTTP/1.1
   |     |     |  Host: node.coap.something.net
   |     |     |  Prefer: long-polling
   |     |     |
   |     +---->|  CON GET
   |     |     |  Uri-Path: temperature
   |     |     |  Observe: 0
   |     |     |
   |     |<----+  ACK 2.05
   |     |     |  Observe: 3482
   |     |     |  "22.1 C"
   |     |     |
   |<----+     |  HTTP/1.1 206 Partial Content
   |     |     |
   |     |     |  22.1 C
   |     |     |
   +---->|     |  GET /temperature HTTP/1.1
   |     |     |  Host: node.coap.something.net
   |     |     |  Prefer: long-polling
   |     |     |
   |     |     |  ... about 60 seconds have passed ...
   |     |     |
   |     |<----+  NON 2.05
   |     |     |  Observe: 3542
   |     |     |  "21.6 C"
   |     |     |
   |<----+     |  HTTP/1.1 206 Partial Content
   |     |     |
   |     |     |  21.6 C
   |     |     |
   +---->|     |  GET /temperature HTTP/1.1
   |     |     |  Host: node.coap.something.net
   |     |     |  Prefer: long-polling
   |     |     |
   |     |     |  ... if the server drops the relationship ...
   |     |     |
   |     |<----+  NON 2.05
   |     |     |  "21.8 C"
   |     |     |
   |<----+     |  HTTP/1.1 200 OK
   |     |     |
   |     |     |  21.8 C
```

Figure 6: HTTP Long Polling to CoAP Observe

Figure 7 shows the interaction between an HTTP client (C), an HC proxy (P), and a CoAP server (S) for the observation of the resource "kitchen/temp" (T) available on S.

It is assumed that the HC proxy knows that the requested resource is observable (since perhaps being asked beforehand to discover its properties as described in Figure 4.) When asked by the HTTP client to retrieve the resource, it requests an observation - in case it weren't already in place - and then sends the collected data to the client as an Atom feed. The data coming through in the constrained network is stored locally on the proxy, and forwarded when further requests are received on the HTTP side. As already said, using the Atom format has two main advantages: first, there is always a "current" feed, but there may also be a complete log made available to HTTP clients; secondly, the HTTP intermediaries can play a substantial role in absorbing a fair amount of the load on the HC proxy. The latter is a very important property when the requested resource is or becomes very popular.

```
 C       P       S
 |       |       |  GET /kitchen/temp HTTP/1.1
 +------>|       |  Host: node.coap.something.net
 |       |       |
 |       +------>|  CON GET
 |       |       |  Uri-Path: kitchen/temp
 |       |       |  Observe: 0
 |       |       |
 |       |<------+  ACK 2.05
 |       |       |  Observe: 1000
 |       |       |  Max-Age: 10
 |       |       |  "22.3 C"
 |       |       |
 |<------+       |  HTTP/1.1 200 OK
 |       |       |  Cache-Control: max-age=10
 |       |       |  ETag: "0x5555"
 |       |       |  Content-Type: application/atom+xml
 |       |       |
 |       |       |  <feed xmlns="http://www.w3.org/2005/Atom">
 |       |       |    <entry>
 |       |       |      <id>urn:uuid:bf08203a-fbbf-49e8-bf11-3c4cff708525</id>
 |       |       |      <updated>2012-03-07T11:14:30</updated>
 |       |       |      <content type="text/plain">
 |       |       |        22.3 C
 |       |       |      </content>
 |       |       |    <entry>
 |       |       |  </feed>
 |       |       |
 |       |       |
 |       |<------+  NON 2.05
 |       |       |  Observe: 1010
 |       |       |  Max-Age: 10
 |       |       |  "22.4 C"
 |       |       |
 +------>|       |  GET /kitchen/temp HTTP/1.1
 |       |       |  Host: node.coap.something.net
 |       |       |
 |       |       |  [...]
 |       |       |
```

Figure 7: Observation via Atom feeds

5. CoAP-HTTP implementation

The CoAP protocol [I-D.ietf-core-coap] allows CoAP clients to request CoAP proxies to perform an HTTP request on their behalf. This is

accomplished by the CoAP client populating an HTTP absolute URI in the 'Proxy-URI' option of the CoAP request to the CoAP proxy. An absolute URI is an HTTP URI that does not contain a fragment component [RFC3986]. The proxy then composes an HTTP request with the given URI and sends it to the appropriate HTTP origin server. The server then returns the HTTP response to the proxy, which the proxy returns to the CoAP client via a CoAP response

5.1. Placement and Deployment

In typical scenarios, for communication from a CoAP client to an HTTP origin server, the HC proxy is expected to be located on the client-side (CS). Specifically, the HC proxy is expected to be deployed at the edge of the constrained network as shown in Figure 8.

The arguments supporting CS placement are as follows:

Client/Proxy/Network configuration overhead: CoAP clients require either static proxy configuration or proxy discovery support. This overhead is simplified if the proxy is placed on the same network domain of the client.

TCP/UDP: Translation between CoAP and HTTP requires also UDP to TCP mapping; UDP performance over the unconstrained Internet may not be adequate. In order to minimize the number of required retransmissions on the constrained part of the network and the overall reliability, TCP/UDP conversion SHOULD be performed as soon as possible in the network path.

Caching: Efficient caching requires that all the CoAP traffic is intercepted by the same proxy, thus a CS placement, collecting all the traffic, is strategic for this need.

Figure 8: Client-side HC proxy deployment scenario

5.2. Basic mapping

The basic mapping of CoAP methods to HTTP is defined in [I-D.ietf-core-coap]. Specifically the {GET, PUT, POST, DELETE} set of CoAP methods are mapped to the equivalent HTTP methods.

In general, an implementation will translate and forward CoAP requests to the HTTP origin server and translate back HTTP responses to CoAP responses, typically employing a certain amount of caching to make this translation more efficient. This section gives some hints for implementing the translation. In addition, some examples are given to illustrate the mappings.

5.2.1. Payloads and Media Types

CoAP supports only a subset of media types. A proxy should convert payloads and approximate content-types as closely as possible. For example, if a HTTP server returns a resource representation in "text/plain; charset=iso-8859-1" format, the proxy should convert the payload to "text/plain; charset=utf-8" format. If conversion is not possible, the proxy can specify a media type of "application/octet-stream".

5.2.2. Max-Age and ETag Options

The proxy can determine the Max-Age Option for responses to GET requests by calculating the freshness lifetime (see Section 13.2.4 of [RFC2616]) of the HTTP resource representation retrieved. The Max-Age Option for responses to POST, PUT or DELETE requests should always be set to 0.

The proxy can assign entity tags to responses it sends to a client. These can be generated locally, if the proxy employs a cache, or be derived from the ETag header field in a response from the HTTP origin server, in which case the proxy can optimize future requests to the HTTP by using Conditional Requests. Note that CoAP does not support weak entity tags.

5.2.3. Use of CoAP blockwise transfer

A CH proxy SHOULD support CoAP blockwise transfers [I-D.ietf-core-block] to allow transport of large CoAP payloads while avoiding link-layer fragmentation in LLNs, and to cope with small datagram buffers in CoAP end-points as described in [I-D.ietf-core-block].

For improved latency a CH proxy MAY initiate a HTTP request triggered by an incoming blockwise CoAP request even when blocks of the CoAP request have only been partially received by the proxy, in cases where the Content-Length field is not going to be used in the HTTP request. This is useful especially if the network between proxy and HTTP server involves low-bandwidth links.

5.2.4. HTTP Status Codes 1xx and 3xx

CoAP does not have provisional responses (HTTP Status Codes 1xx) or responses indicating that further action needs to be taken (HTTP Status Codes 3xx). When a proxy receives such a response from the HTTP server, the response should cause the proxy to complete the request, for example, by following redirects. If the proxy is unable or unwilling to do so, it can return a 5.02 (Bad Gateway) error.

5.2.5. Examples

Figure 9 shows an example implementation of a basic CoAP GET request with an HTTP URI as the value of a Proxy-URI option. The proxy retrieves a representation of the target resource from the HTTP origin server. It converts the payload to a UTF-8 charset, calculates the Max-Age Option from the Expires header field, and derives an entity-tag from the ETag header field.

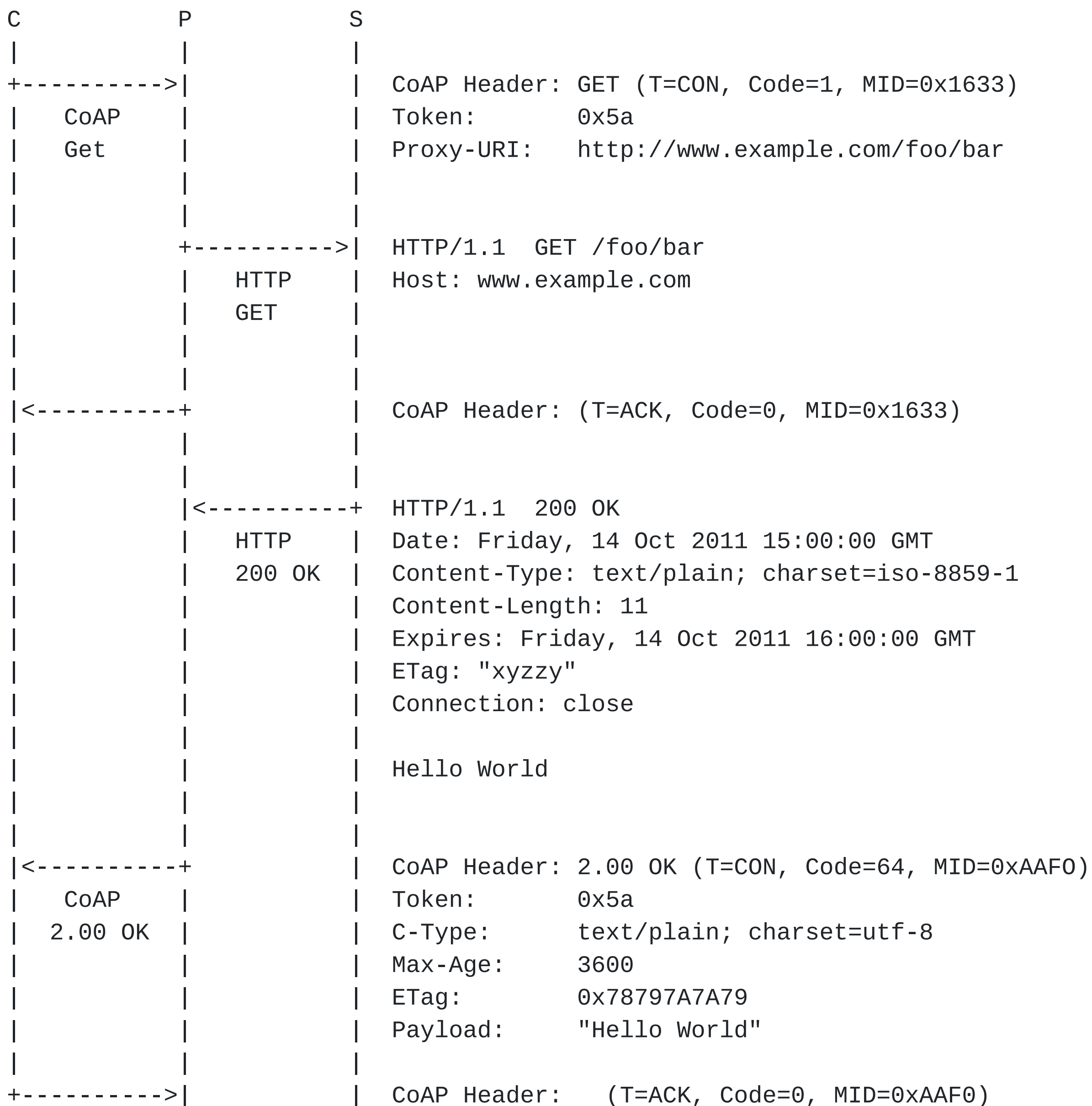

Figure 9: A basic CoAP-HTTP GET request

The example in Figure 10 builds on the previous example and shows an implementation of a GET request that includes a previously returned ETag Option. The proxy makes a Conditional Request to the HTTP origin server by including an If-None-Match header field in the HTTP GET Request. The CoAP response indicates that the response stored by the client is fresh. It includes a Max-Age Option calculated from the HTTP response's Expires header field.

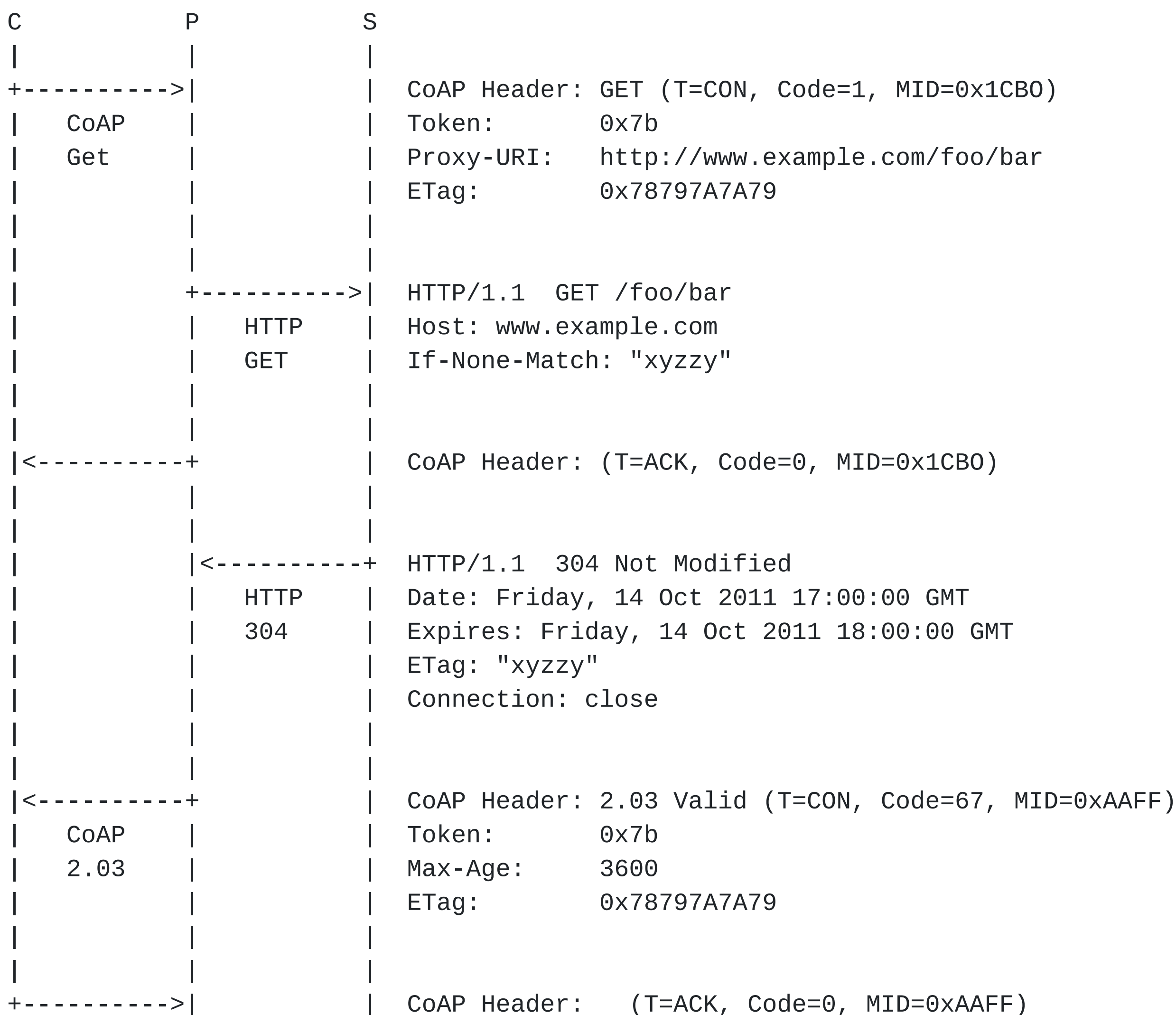

Figure 10: A CoAP-HTTP GET request with an ETag Option

6. Security Considerations

The security concerns raised in Section 15.7 of [RFC2616] also apply to the HC proxy scenario. In fact, the HC proxy is a trusted (not rarely a transparently trusted) component in the network path.

The trustworthiness assumption on the HC proxy cannot be dropped. Even if we had a blind, bi-directional, end-to-end, tunneling facility like the one provided by the CONNECT method in HTTP, and also assuming the existence of a DTLS-TLS transparent mapping, the two tunneled ends should be speaking the same application protocol, which is not the case. Basically, the protocol translation function is a core duty of the HC proxy that can't be removed, and makes it a

necessarily trusted, impossible to bypass, component in the communication path.

A reverse proxy deployed at the boundary of a constrained network is an easy single point of failure for reducing availability. As such, a special care should be taken in designing, developing and operating it, keeping in mind that, in most cases, it could have fewer limitations than the constrained devices it is serving.

The following sub paragraphs categorize and argue about a set of specific security issues related to the translation, caching and forwarding functionality exposed by an HC proxy module.

6.1. Traffic overflow

Due to the typically constrained nature of CoAP nodes, particular attention SHOULD be posed in the implementation of traffic reduction mechanisms (see Section 4.2.1), because inefficient implementations can be targeted by unconstrained Internet attackers. Bandwidth or complexity involved in such attacks is very low.

An amplification attack to the constrained network may be triggered by a multicast request generated by a single HTTP request mapped to a CoAP multicast resource, as considered in Section XX of [I-D.ietf-core-coap].

The impact of this amplification technique is higher than an amplification attack carried out by a malicious constrained device (i.e. ICMPv6 flooding, like Packet Too Big, or Parameter Problem on a multicast destination [RFC4732]), since it does not require direct access to the constrained network.

The feasibility of this attack, disruptive in terms of CoAP server availability, can be limited by access controlling the exposed HTTP multicast resource, so that only known/authorized users access such URIs.

6.2. Cross-protocol security policy mapping

At the moment of this writing, CoAP and HTTP are missing any cross-protocol security policy mapping.

The HC proxy SHOULD flexibly support security policies between the two protocols, possibly as part of the HC URI mapping function, in order to statically map HTTP and CoAP security policies at the proxy (see Appendix A.2 for an example.)

6.3. Handling secured exchanges

It is possible that the request from the client to the HC proxy is sent over a secured connection. However, there may or may not exist a secure connection mapping to the other protocol. For example, a secure distribution method for multicast traffic is complex and MAY not be implemented (see [I-D.ietf-core-groupcomm]).

By default, an HC proxy SHOULD reject any secured client request if there is no configured security policy mapping. This recommendation MAY be relaxed in case the destination network is believed to be secured by other, complementary, means. E.g.: assumed that CoAP nodes are isolated behind a firewall (e.g. as the SS HC proxy deployment shown in Figure 1), the HC proxy may be configured to translate the incoming HTTPS request using plain CoAP (i.e. NoSec mode.)

The HC URI mapping MUST NOT map to HTTP (see Section 3.1) a CoAP resource intended to be accessed only using HTTPS.

A secured connection that is terminated at the HC proxy, i.e. the proxy decrypts secured data locally, raises an ambiguity about the cacheability of the requested resource. The HC proxy SHOULD NOT cache any secured content to avoid any leak of secured information. However in some specific scenario, a security/efficiency trade-off could motivate caching secured information; in that case the caching behavior MAY be tuned to some extent on a per-resource basis (see Section 6.2).

6.4. Spoofing and Cache Poisoning

In web security jargon, the "cache poisoning" verb accounts for attacks where an evil user causes the proxy server to associate incorrect content to a cached resource, which work through especially crafted HTTP requests or request/response combos.

When working in CoAP NoSec mode, the use of UDP makes cache poisoning on the constrained network easy and effective, simple address spoofing by a malicious host is sufficient to perform the attack. The implicit broadcast nature of typical link-layer communication technologies used in constrained networks lead this attack to be easily performed by any host, even without the requirement of being a router in the network. The ultimate outcome depends on both the order of arrival of packets (legitimate and rogue) and the processing/discarding policy at the CoAP node; attackers targeting this weakness may have less requirements on timing, thus leading the attack to succeed with high probability.

In case the threat of a rogue mote acting in the constrained network can't be winded up by appropriate procedural means, the only way to avoid such attacks is for any CoAP server to work at least in MultiKey mode with a 1:1 key with the HC proxy. SharedKey mode would just mitigate the attack, since a guessable MIDs and Tokens generation function at the HC proxy side would make it feasible for the evil mote to implement a "try until succeed" strategy. Also, (authenticated) encryption at a lower layer (MAC/PHY) could be defeated by a slightly more powerful attacker, a compromised router mote.

6.5. Subscription

As noted in Section 7 of [I-D.ietf-core-observe], when using the observe pattern, an attacker could easily impose resource exhaustion on a naive server who's indiscriminately accepting observer relationships establishment from clients. The converse of this problem is also present, a malicious client may also target the HC proxy itself, by trying to exhaust the HTTP connection limit of the proxy by opening multiple subscriptions to some CoAP resource.

Effective strategies to reduce success of such a DoS on the HTTP side (by forcing prior identification of the HTTP client via usual web authentication mechanisms), must always be weighted against an acceptable level of usability of the exposed CoAP resources.

7. Acknowledgements

Special credit is given to Klaus Hartke who provided the text for Section 5 and a lot of direct input to this document. Special credit about the text in Section 5 is given to Carsten Bormann who provied parts of it.

Thanks to Zach Shelby, Michele Rossi, Nicola Bui, Michele Zorzi, Peter Saint-Andre, Cullen Jennings, Kepeng Li, Brian Frank, Peter Van Der Stok, Kerry Lynn, Linyi Tian, Dorothy Gellert for helpful comments and discussions that have shaped the document.

8. References

8.1. Normative References

[I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-04 (work in progress), July 2011.

[I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-07 (work in progress), July 2011.

[I-D.ietf-core-groupcomm]
Rahman, A. and E. Dijk, "Group Communication for CoAP", draft-ietf-core-groupcomm-00 (work in progress), January 2012.

[I-D.ietf-core-observe]
Hartke, K. and Z. Shelby, "Observing Resources in CoAP", draft-ietf-core-observe-02 (work in progress), March 2011.

[I-D.ietf-httpbis-p1-messaging]
Fielding, R., Gettys, J., Mogul, J., Nielsen, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., and J. Reschke, "HTTP/1.1, part 1: URIs, Connections, and Message Parsing", draft-ietf-httpbis-p1-messaging-18 (work in progress), January 2012.

[I-D.ietf-httpbis-p2-semantics]
Fielding, R., Gettys, J., Mogul, J., Nielsen, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., and J. Reschke, "HTTP/1.1, part 2: Message Semantics", draft-ietf-httpbis-p2-semantics-18 (work in progress), January 2012.

[I-D.thomson-hybi-http-timeout]
Thomson, M., Loreto, S., and G. Wilkins, "Hypertext Transfer Protocol (HTTP) Timeouts", draft-thomson-hybi-http-timeout-00 (work in progress), March 2011.

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4287] Nottingham, M., Ed. and R. Sayre, Ed., "The Atom Syndication Format", RFC 4287, December 2005.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.

8.2. Informative References

[I-D.bormann-core-simple-server-discovery]
Bormann, C., "CoRE Simple Server Discovery", draft-bormann-core-simple-server-discovery-00 (work in progress), March 2011.

[I-D.eggert-core-congestion-control]
Eggert, L., "Congestion Control for the Constrained Application Protocol (CoAP)", draft-eggert-core-congestion-control-01 (work in progress), January 2011.

[I-D.shelby-core-resource-directory]
Shelby, Z. and S. Krco, "CoRE Resource Directory", draft-shelby-core-resource-directory-01 (work in progress), September 2011.

[I-D.snell-http-prefer]
Snell, J., "Prefer Header for HTTP", draft-snell-http-prefer-12 (work in progress), February 2012.

[I-D.vanderstok-core-bc]
Stok, P. and K. Lynn, "CoAP Utilization for Building Control", draft-vanderstok-core-bc-04 (work in progress), July 2011.

[RFC3040] Cooper, I., Melve, I., and G. Tomlinson, "Internet Web Replication and Caching Taxonomy", RFC 3040, January 2001.

[RFC4732] Handley, M., Rescorla, E., and IAB, "Internet Denial-of-Service Considerations", RFC 4732, December 2006.

[RFC6202] Loreto, S., Saint-Andre, P., Salsano, S., and G. Wilkins, "Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP", RFC 6202, April 2011.

[W3C.HTML5]
Hickson, I., "HTML5", World Wide Web Consortium WD (work in progress) WD-html5-20111018, October 2011, <http://dev.w3.org/html5/spec/>.

Appendix A. Internal Mapping Functions (from an implementer's perspective)

At least three mapping functions have been identified, which take place at different stages of the HC proxy processing chain, involving the URL, Content-Type and Security Policy translation.

All these maps are required to have at least URL granularity so that, in principle, each and every requested URL may be treated as an independent mapping source.

In the following, the said map functions are characterized via their expected input and output, and a simple, yet sufficiently rich, configuration syntax is suggested.

In the spirit of a document providing implementation guidance, the specification of a map grammar aims at putting the basis for a reusable software component (e.g. a stand-alone C library) that many different proxy implementations can link to, and benefit from.

A.1. URL Map Algorithm

In case the HC proxy is a reverse proxy, i.e. it acts as the origin server in face of the served network, the URL of the resource requested by its clients (perhaps having an 'http' scheme) shall be mapped to the real resource origin (perhaps in the 'coap' scheme).

In case HC is a forward proxy, no URL translation is needed since the client already knows the "real name" of the resource.

An interception HC proxy, instead, MAY use the homogeneous mapping strategy (see Section 3.1.1 for details) to operate without any pre-configuration need.

As noted in Appendix B of [RFC3986] any correctly formatted URL can be matched by a POSIX regular expression. By leveraging on this property, we suggest a syntax that describes the URL mapping in terms of substituting the regex-matching portions of the requested URL into the mapped URL template.

E.g.: given the source regular expression '^http://example.com/coap/.*$' and destination template 'coap://$1' (where $1 stands for the first - and only in this specific case - substring matched by the regex pattern in the source), the input URL "http://example.com/coap/node1/resource2" translates to "coap://node1/resource2".

This is a well established technique used in many todays web

components (e.g. Django URL dispatcher, Apache mod_rewrite, etc.), which provides a compact and powerful engine to implement what essentially is an URL rewrite function.

```
INPUT
    * requested URL

OUTPUT
    * target URL

SYNTAX
    url_map [rule name] {
        requested_url   <regex>
        mapped_url      <regex match subst template>
    }

EXAMPLE 1
    url_map homogeneous {
        requested_url   '^http://.*$'
        mapped_url      'coap//$1'
    }

EXAMPLE 2
    url_map embedded {
        requested_url   '^http://example.com/coap/.*$'
        mapped_url      'coap//$1'
    }
```

Note that many different url_map records may be given in order to build the whole mapping function. Each of these records can be queried (in some predefined order) by the HC proxy until a match is found, or the list is exhausted. In the latter case, depending on the mapping policy (only internal, internal then external, etc.) the original request can be refused, or the same mapping query is forwarded to one or more external URL mapping components.

A.2. Security Policy Map Algorithm

In case the "incoming" URL has been successfully translated, the HC proxy must lookup the security policy, if any, that needs to be applied to the request/response transaction carried on the "outgoing" leg.

```
INPUT
    * target URL (after URL map has been applied)
    * original requester identity (given by cookie, or IP address, or
      crypto credentials/security context, etc.)

OUTPUT
    * security context that will be applied to access the target URL

SYNTAX
    sec_map [rule name] {
        target_url      <regex>     -- one or more
        requester_id    [TBD]
        sec_context     [TBD]
    }

EXAMPLE
    [TBD]
```

A.3. Content-Type Map Algorithm

In case a set of destination URLs is known as being limited in handling a narrow subset of mime types, a content-type map can be configured in order to let the HC proxy transparently handle the compatible/lossless format translation.

```
INPUT
    * destination URL (after URL map has been applied)
    * original content-type

OUTPUT
    * mapped content-type

SYNTAX
    ct_map {
        target_url  <regex>                 -- one or more targetURLs
        ct_switch   <source_ct, dest_ct>    -- one or more CTs
    }

EXAMPLE
    ct_map {
        target_url  '^coap://class-1-device/.*$'
        ct_switch   */xml   application/exi
    }
```

Authors' Addresses

Angelo P. Castellani
University of Padova
Via Gradenigo 6/B
Padova 35131
Italy

Email: angelo@castellani.net

Salvatore Loreto
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: salvatore.loreto@ericsson.com

Akbar Rahman
InterDigital Communications, LLC

Email: Akbar.Rahman@InterDigital.com

Thomas Fossati
KoanLogic
Via di Sabbiuno 11/5
Bologna 40136
Italy

Phone: +39 051 644 82 68
Email: tho@koanlogic.com

Esko Dijk
Philips Research

Email: esko.dijk@philips.com